Certifying Failed Lean Proof-Agent Searches with Exact Replay

Ryan Farell [1] **Chandrajit L. Bajaj** [1]

Abstract

Large language models can propose Lean tactics (proof steps that Lean accepts or rejects), but a failed proof search is rarely actionable for a mathematical agent: did it search the proposed tactics poorly, miss a key tactic, or attempt an out-of-reach goal? We make one part of this diagnosis exact. At each reached Lean state we save the tactics the model proposed — its *recorded support* — and replay only those saved tactics under fixed depth and timeout bounds. Replay returns a Lean-checked proof, a *closed certificate* that no saved tactic path proves the goal within those bounds, or a timeout transcript. The closed certificate is the useful signal: it shows that replaying the same tactics cannot help, so any repair must add new candidates. We evaluate on CSLib-Holes, a frozen suite of 100 mechanically selected proof holes from the public CSLib Lean project, with the original proofs hidden from the agent. Base BFS-Prover replay solves 10/100 and leaves 74 closed certificates. Adding the statements of in-scope lemmas to the prompt repairs 16/74, against 6/74 for a matched search-only widening, and iterating these Lean-checked additions raises the oracle-free solved count from 10/100 to 39/100, every repair re-checked by Lean. A target-statement control repairs 12/74, so whether the failed-search frontier (the goals left unproved at the dead ends of the failed search) helps beyond the target statement is directional rather than decisive (pooled exact McNemar $p = 0.227$). This is a bounded audit layer for proof agents.

1. Introduction

Formal theorem proving is an attractive test bed for mathematical agents because every proposed step can be checked by a proof assistant. A *tactic* is a Lean proof step, and a *proof state* is the current list of goals and hypotheses. A Lean agent can ask a language model for candidate tactics, apply them to the current proof state, and keep only the transitions Lean accepts (de Moura & Ullrich, 2021). This gives a verifier loop, but it does not by itself explain failure. When the agent fails to prove a goal, did it search the exposed tactics poorly, omit a critical tactic from the sampled support, or attempt a theorem that is false?

This paper isolates the part of that question that can be answered exactly. We treat the recorded tactic candidates for each reachable Lean proof state as a finite support. Within a fixed Lean version, imports, candidate budget, and depth bound, exact replay can prove a bounded statement: either there is a Lean-accepted proof path in the recorded support, or there is not. When the graph closes — every proof path the saved candidates can express within the fixed bounds has been tried, and none is accepted by Lean — the negative outcome is a *closed certificate*. This is deliberately weaker than theorem unprovability and weaker than a claim about the model's full distribution, but it is stronger than another failed validator loop over the same candidates.

The contribution is operational. A certificate identifies states where the recorded support is exhausted under the fixed bounds, and we then test fixed policies that add new candidates at those states. In CSLib-Holes, those policies retrieve in-scope lemma signatures or apply deterministic tactics built from them. A candidate update counts only when the resulting proof replays in Lean. We do not update model weights or claim autonomous theorem discovery.

Figure 1 shows the loop we evaluate. The agent does not trust the model's proof text. It stores the recorded support, replays it through Lean, and then chooses a next action only from Lean-checked outcomes: a checked proof, a closed certificate, or a resource-bound transcript.

We make three contributions.

- We formulate the tactics a model proposes at each Lean state as a finite support and state exactly what replay guarantees: replay returns a Lean-checked proof when it finds an accepted path, a closed certificate when the finite recorded support is exhausted under fixed depth and timeout settings, or a resource-bound transcript

[1]The University of Texas at Austin, Austin, TX, USA. Correspondence to: Ryan Farell <ryan.farell@gmail.com>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

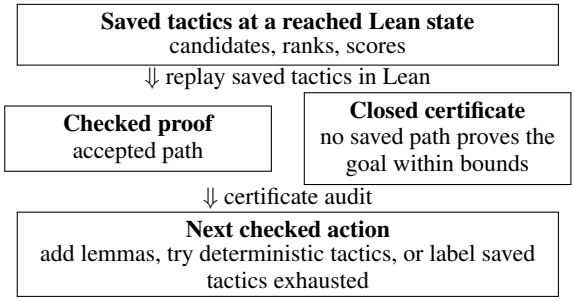

Figure 1. The audit loop replays the saved tactics in Lean: proof and repair steps are Lean-checked, a closed certificate marks the saved tactics exhausted within the fixed bounds, and timeout outcomes are logged as resource-bound transcripts.

when timeouts prevent closure.

- We implement a reproducible Lean REPL pipeline that records tactic candidates, Lean transitions, verifier calls, timeout metadata, exact score pruning, and replay repairs. Every accepted transition is checked by Lean, and `sorry`, `admit`, and unchecked proof artifacts are rejected.
- We introduce CSLib-Holes, a 100-case suite mechanically derived from the public CSLib Lean project.[1] For cases with a closed certificate, we retrieve the statements of in-scope Lean lemmas and add them to the prompt (the agent never sees the original proofs), rechecking every repair in Lean. Iterated Lean-checked support updates raise the oracle-free closure from 10/100 to 39/100; the full per-arm comparison and frontier analysis are reported in the Results.

One CSLib-Holes case makes the loop concrete. For `mergesort_time`, a running-time bound for merge sort, exact replay closes the saved BFS-Prover tactics without a Lean-accepted proof, so the case becomes one of the 74 closed certificates. A matched search-only widening of the same support still fails. Retrieval then surfaces an in-scope sibling lemma, the model assembles a short proof from it, and Lean accepts the repair. A target-statement control does not repair this case. The certificate does not say the theorem is unprovable, only that the saved tactics are exhausted under the fixed bounds, so any repair must add new candidates and still replay in Lean. The rest of the paper formalizes this loop and reports how many of the 100 CSLib-Holes admit similar audit-driven repairs.

2. Finite-Support Lean Replay

Proof-state graph. For each theorem x, the Lean REPL returns an initial proof state s_0 with a goal stack. At any reached state s, a model prompt produces an ordered finite candidate set

$$C_b(s) = \{(t_i, \ell_i, r_i)\}_{i=1}^{b},$$

where t_i is a tactic string, ℓ_i is its conditional log-probability under the model, and r_i is its rank in the recorded support. Querying Lean defines a partial transition

$$\delta(s, t) \in \{\text{FAIL}, \text{SOLVED}\} \cup S.$$

Only transitions with a new Lean proof state and no Lean error are retained. Any candidate containing unsound markers such as `sorry` or `admit` is rejected by the runner.

Exact replay. Let $\Pi_b(x)$ be all tactic paths reachable from s_0 using only candidates in the recorded supports $C_b(s)$ and a fixed depth bound. A path π has score $L(\pi) = \sum_{t_i \in \pi} \ell_i$. Exact replay returns the highest-scoring path accepted by Lean:

$$\pi^\star = \arg \max_{\pi \in \Pi_b(x):\ \text{Lean}(\pi)=\text{SOLVED}} L(\pi).$$

If no such path exists after closing the finite reachable graph, replay emits a certificate containing the theorem id, Lean configuration, candidate supports, reachable states, failed transitions, and bounds. In the final BFS-Prover runs we use an exact branch-and-bound variant: once a completed proof with score $L^\star$ is found, any partial path scoring at most $L^\star$ is pruned: each $\ell_i \leq 0$, so extending a path cannot raise its score. This preserves the exact optimum within the recorded support while reducing verifier calls on feasible cases.

Proposition 1 (Certificate scope)**.** *If exact replay returns a certificate for theorem x under support C_b and bound d, then no path in the recorded graph $\Pi_b(x)$ of depth at most d is accepted by the fixed Lean verifier.*

The proposition is intentionally finite. It does not imply that x is false, that Lean lacks a proof, or that the model assigned zero probability to a proof outside the acquired support. When a tactic, command, or case exceeds the configured resource limit, the runner logs a timeout transition. A negative artifact containing such transitions is exact only for that bounded evaluator and timeout policy. It is not a claim about unbounded Lean elaboration. We therefore reserve the certificate guarantee for the closed finite replay graph and treat timeout rows as resource-bound transcripts.

Verifier-checked repair. We add deterministic repair candidates only when they can be generated from the current Lean goal/context and the model's existing candidate row. The repairs currently cover conjunction projection and reassociation, two-step `Iff.trans`, two-step equality transitivity with symmetry, and direct universal/existential projection from conjunction witnesses. Each repair is logged as an extra candidate with source `structural_repair`; failed repairs still count as Lean verifier calls.

[1] Code, benchmark, and audit checks: https://github.com/rfarell/cslib-holes.

3. Experiment

Task suite. CSLib-Holes is the primary experiment. CSLib is the theoretical-computer-science Lean library used by the AI4Math Track 2 challenge, which makes it a workshop-native substrate for testing audit semantics. We use it to build a benchmark of our own, not as a challenge entry. We check out the public CSLib Lean project at commit `caa9a9c1fad8d6ae627941ae921557731da055d7` with toolchain `leanprover/lean4:v4.30.0-rc2`. We mechanically scan `Cslib/` declarations in lexicographic source-path order and then source line order. Eligible declarations are `theorem` or `lemma` commands with a `:= by` proof, no `sorry` or `admit`, one to 40 nonblank proof lines, and at most 120K command characters. We take the first 100 eligible declarations after this filter. For each declaration, we replace the proof body by a proof hole and store the original proof only in a separate oracle file. The frozen manifest records source path, line, Lean command, prefix environment, target command, and original proof script. The generator never uses oracle proofs as candidates.

All 100 generated holes create Lean proof states. Original CSLib proof replay is stricter because extracted proof bodies can depend on local syntax, scoping, and multi-command context; 23 cases form a conservative proof-state oracle-clean subset where the original CSLib proof replays successfully through the same tactic REPL interface. We also run a command-level oracle replay that restores each stored proof body into the original declaration and checks it with Lean. This broader command-level diagnostic covers 87/100 holes and 81 base negative artifacts. We use it only for post-hoc taxonomy; no oracle proof body enters decoding, retrieval, support acquisition, or repair.

We retain earlier curated Lean proof-state suites only as sanity checks for the audit machinery. They are useful for verifying exact replay, structural repair, false controls, and branch-and-bound pruning, but they are not the main external-validity claim.

Model and verifier. Candidate acquisition uses cached Qwen2.5-7B-Instruct (Qwen Team, 2024) and BFS-Prover-V2-7B (Xin et al., 2025) on A100 GPUs. Lean verification uses a repo-local `leanprover-community/repl` build and Lean 4.30.0-rc2. BFS-Prover uses its proof-state separator prompt; Qwen uses a short instruction prompt that asks for one Lean tactic. Every reported proof is replayed through Lean. Every negative artifact records the saved candidate support, depth, timeout policy, and Lean configuration that bound the claim.

Baselines and metrics. We report a fixed static tactic baseline, raw exact replay over model support, and deterministic union/repair over the same support. The main support-update experiment is oracle-free: we take the 74 full-set BFS-Prover negatives whose finite replay graphs close without timeout and rerun only those certificate cases under matched beam, depth, timeout, and token settings. The no-retrieval control widens support with the same prompt family but no additional retrieved declarations. The target-only retrieval control ranks declarations from the theorem statement and source metadata only, with no information from the failed search. The frontier-retrieval arm appends distinct open goals observed in the closed-certificate trace to the same case-level retrieval query before ranking signatures. Both arms build one ranked signature set per case and add it to the prompt for the whole case. The frontier arm pools those goals rather than aiming retrieval at a particular leaf or proof state, so it is a frontier-conditioned support-update arm, not per-state localization. We also report deterministic tactics over the same retrieved signatures and compose all Lean-checked repairs back into the fixed 100-case base run. Budget-matched here means matched case set, beam, depth, timeout, and token settings. Realized verifier calls can differ because the reachable graphs differ.

Retrieval is automatic and fixed before evaluating the audited support gaps. For each case, the retrieval corpus consists of declarations from imported CSLib modules plus declarations strictly before the target declaration in the per-case source prefix. We expose only declaration names, namespace/module information, and type signatures. The target declaration itself, proof bodies, original proof scripts, oracle files, and same-file post-target declarations are excluded from the retrieval corpus and from candidate generation. The deterministic ranker scores token overlap between the target command and candidate signatures, with a small declaration name and source-recency boost, and returns the top 12 signatures under a 2000 character per-signature cap.

Metrics are solved cases, negative artifacts, success rate, mean Lean verifier calls, and oracle-gap taxonomy. Wall-clock and timeout metadata are recorded in the artifacts; the main table uses verifier calls as the compact audit trace statistic. Oracle scripts are diagnostic only: they classify certificates after the run and are not deployable decoder candidates.

4. Results

Table 1 reports the audit in four stages: base replay and the diagnostic oracle-clean subset; the oracle-free support updates triggered by the 74 closed certificates; the iterated residual rounds; and the composed closures in the fixed 100-case base run. The headline progression is 10/100 base replay $\rightarrow$ 28/100 after the first composed pass $\rightarrow$ 38/100 frontier/control composition $\rightarrow$ 39/100 legal oracle-free portfolio. On the full 100-case CSLib-Holes benchmark, BFS-Prover exact replay solves 10 cases and emits 90 nega-

Table 1. CSLib-Holes external proof-hole results. Rows progress from base replay, to oracle-free support updates on the 74 closed certificates, to iterated residual rounds, to composed closures in the fixed 100-case base run. Base replay solves 10/100 and the composed portfolio reaches 39/100, every counted closure Lean-checked. The 74-case rows rerun exactly the closed finite-support certificates from the BFS-Prover base run, with no oracle proof body used to choose these triggers. Closure rows compose independently Lean-replayed repairs into the fixed base run, not fresh reruns over every full-100 negative artifact. Neg. counts unsolved artifacts: closed finite replays are certificates, while artifacts with timeout transitions are resource-bound transcripts. The target-only row accounts for most of the Round 5 frontier gain (the strongest retrieval baseline), and the legal (oracle-free, Lean-checked) portfolio row adds its one remaining checked success to the previous frontier/control composition. Mean Lean calls are audit-trace statistics, not total acquisition-cost comparisons for union or timeout rows.

Method	Scope	Cases	Solved	Neg.	Rate	Mean Lean calls
Base replay and diagnostic oracle-clean subset						
Static tactics	oracle-clean 23	23	3	20	.130	9.00
Qwen2.5-7B exact	oracle-clean 23	23	0	23	.000	6.87
BFS-Prover exact	full 100	100	10	90	.100	43.56
Automation menu (12 tactics)	full 100	100	6	94	.060	11.59
BFS-Prover exact	oracle-clean 23	23	6	17	.261	53.96
BFS-Prover union repair	oracle-clean 23	23	6	17	.261	65.43
Oracle-free support updates on closed base certificates						
No-retrieval wider replay	closed cert. 74	74	6	68	.081	1248.22
Certificate-frontier retrieval	closed cert. 74	74	16	58	.216	1401.05
Target-only retrieval control	closed cert. 74	74	12	62	.162	1144.54
Deterministic frontier tactics	closed cert. 74	74	7	67	.095	82.50
Frontier support union	success union	18	18	0	1.000	426.00
Iterated residual support expansion						
Round 6 frontier retrieval	residual 72	72	5	67	.069	1194.29
Round 6 no-retrieval control	residual 72	72	4	68	.056	1158.65
Round 7 frontier retrieval	residual 67	67	3	64	.045	1180.51
Round 7 no-retrieval control	residual 67	67	2	65	.030	1064.67
Round 8 frontier retrieval	residual 62	62	0	62	.000	1328.23
Round 8 no-retrieval control	residual 62	62	0	62	.000	1040.74
Composed closures in the fixed base run						
No-retrieval closure	full 100	100	16	84	.160	117.18
Frontier-model closure	full 100	100	26	74	.260	105.34
Frontier-support union closure	full 100	100	28	72	.280	104.95
Round 6 frontier closure	full 100	100	33	67	.330	144.48
Round 6 frontier+control closure	full 100	100	35	65	.350	201.09
Round 7 frontier closure	full 100	100	36	64	.360	152.18
Round 7 frontier/control closure	full 100	100	38	62	.380	208.79
Legal portfolio closure	full 100	100	39	61	.390	213.17

tive artifacts. These artifacts are not support-gap labels and are not claims that the underlying CSLib theorems are false. Each records the bounded replay outcome under the saved candidate support, depth bound, Lean version, and timeout policy; if a timeout transition is present, the artifact is a resource-bound transcript rather than a closed finite-support certificate. Among the 90 negatives, 74 are closed finite-support certificates. These 74 cases are the main trigger set for support acquisition because they are selected from the replay artifact itself, not from oracle proof bodies.

The oracle-clean evidence remains diagnostic. The proof-state subset contains the 23 cases whose original CSLib proof bodies replay through the same tactic REPL interface. BFS-Prover solves 6/23, and the oracle-gap audit labels the 17 remaining proof-state oracle-clean certificates as exhausted recorded support rather than intentionally unprovable cases or oracle replay failures. A broader command-level replay, which restores stored proof bodies into their original declarations and asks Lean to check the commands, covers 87/100 holes and 81 base negative artifacts. This broadens the post-hoc taxonomy but remains separate from the oracle-free support-update experiment, whose trigger set is the larger set of 74 closed base certificates selected from the replay artifact itself.

The base and diagnostic baselines expose the difficulty of this external slice. A fixed static tactic menu solves 3/23 oracle-clean cases. Qwen2.5-7B under the same finite replay protocol solves none of the oracle-clean cases. Deterministic union/repair over the saved BFS-Prover support does not add closures, so the remaining failures are not simple instances of the current repair templates.

The main support-update result is a direct comparison on the same 74 closed base certificates. A no-retrieval wider replay repairs 6/74. Target-only retrieval, using the same model/search budget but retrieving from the target statement rather than failed-search frontier states, repairs 12/74. Automatic certificate-frontier retrieval repairs 16/74 under the same BFS-Prover model, beam, depth, token, timeout, and score-pruning settings. The frontier-vs-target paired comparison has 10 both-solved, 6 frontier-only, 2 target-only, and 56 neither cases, with exact McNemar $p = 0.2891$. This is a modest directional

edge for frontier retrieval, not evidence that frontier context uniquely explains the Round 5 gain. The frontier-only repairs include mergesort_time, reindex_run_iff, lts_mem_pairvialang, le_omegapow_congr, congr_language_eq, and several omega-language algebra facts. The target-only arm has two cases not repaired by the frontier arm, sorted_merge and hmul_bot, so target-only retrieval is the strongest retrieval baseline.

Across the 74 certificates, the aggregate result follows the same pattern as the mergesort_time trace from the introduction.

We also test a practical baseline that does not call a model after retrieval. It applies a fixed tactic menu to the retrieved declaration names: simp, simp_all, grind, rw, direct exact, and simple fun_induction variants over the top-12 retrieved signatures. This deterministic frontier baseline repairs 7/74. It overlaps with the model arm on five cases and adds concat_start_right and lts_pairlang_split. Taking the union of deterministic and BFS-Prover frontier-retrieval successes gives 18 distinct support updates, each replayed by Lean.

The closure rows show the end-to-end effect in the fixed base run. The no-retrieval support updates raise the full 100-case closure from 10/100 to 16/100. The frontier model alone raises it to 26/100, and the union with deterministic frontier tactics raises it to 28/100, replacing 18 closed base certificates with Lean-accepted proofs while leaving the original base successes unchanged.

Iterating the same policy over the residual certificates adds verified repairs in two more rounds. A second pass over the 72 remaining cases repairs 5/72, and the matched no-retrieval control repairs 4/72 on different cases, so their composition reaches 35/100. A third pass over the 67 frontier-residual cases repairs zero_hmul, mem_ssup, and isup_hmul, lifting the frontier-only closure to 36/100, while the union with the completed Round 6 no-retrieval control reaches 38/100.

Past this point the rounds saturate. The completed Round 7 no-retrieval control repairs two different cases, sorted_merge and mul_sub_one, both already covered by the earlier Round 6 no-retrieval union. A paired comparison over the 67 shared residual cases gives 0 both-solved, 3 frontier-only, 2 no-retrieval-only, and 62 neither, so the Round 7 frontier repairs are not repeated no-context wider-search wins. A fourth pass over the 62 remaining cases repairs 0/62 under both the frontier and the matched no-retrieval policies, with no discordant successes. The strongest frontier/control composition therefore settles at 38/100, and adding the one target-only success not already covered, hmul_bot, gives the current legal oracle-free portfolio closure of 39/100. Because Round 7 contributes only repairs already inside that 38/100 composition and Round 8 adds none under either checked policy, 39/100 is the portfolio baseline any future primary improvement should be scored against.

This is the result the audit loop is designed to enable: a closed certificate identifies exhausted recorded support, a fixed support-acquisition policy changes that support, and Lean replay decides whether the update was a real repair.

Three checks bound how to read that 39/100 portfolio: whether simple automation explains it, whether the pattern survives on a frozen held-out set, and what the certificate frontier does and does not prove.

A fixed automation menu of twelve standard Lean tactics (simp_all, aesop, exact?, grind, and related tactics) applied directly to each hole solves only 6/100 under the same per-tactic timeout, and every one of those six is already inside the 39/100 portfolio. This twelve-tactic automation menu therefore does not by itself reach the reported closures. The base BFS-Prover replay alone exceeds it.

Held-out external validation. To test whether the development closure reflects the method rather than adaptive tuning on the same hundred holes, we froze the entire pipeline (model, budgets, retrieval ranker, repair menu, and round structure) and ran it once, with no tuning, on a mechanically generated held-out second hundred (CSLib-Holes 101–200, frozen before any model run). Base BFS-Prover replay solves 20/99 and the composed pipeline closes 34/99 (rates over the 99 runnable held-out holes, with one documented generation exclusion), against 10/100 and 39/100 on the development set: the composed closure broadly reproduces on a frozen, untuned second hundred, a sanity check against test-set adaptivity rather than a separate validation benchmark. Among the 79 held-out cases that close under base replay, the retrieval-vs-no-retrieval ordering also reproduces (frontier 6, target-only 7, deterministic 8, versus 4 for no-retrieval wider replay). Holes 101–200 include heavier files whose Lean elaboration is slower, leaving 10–14 cases per arm resource-bound at the frozen budget; these are counted as non-repairs, so 34/99 is a conservative lower bound. For the held-out frontier-vs-target significance test, the paired set is the 63 certificates with both frontier and target result rows (69 frontier rows and 70 target rows before pairing).

Frontier versus target, pooled. Pooling the 74 development paired rows with those 63 held-out paired rows (137 paired rows total) gives 8 frontier-only and 3 target-only discordant repairs (exact McNemar $p = 0.227$); the held-out half alone is 2 versus 1 ($p = 1.0$). Adding the held-out cases did not resolve the frontier-versus-target comparison, so we report it as a directional rather than decisive effect: retrieval of either kind clearly helps over no retrieval, but

the present evidence does not establish that failed-search frontier states are uniquely responsible for the gain.

Trigger soundness and cost. Two controls further characterize the audit layer. First, running the same acquisition arms on the 16 resource-bound timeout transcripts repairs them at a rate indistinguishable from the closed certificates (no-retrieval 2/16 vs 6/74, exact $p = 0.63$; target-only 2/16 vs 12/74, $p = 1.0$), and frontier retrieval is undefined on a transcript that never closed. The closed certificate is thus the trigger set because it is a sound, exhausted-support object with the finite-replay guarantee, not because it is differentially more repairable. Second, per verified repair, certificate-frontier retrieval uses about 6.5k Lean calls versus about 15k for no-retrieval wider replay. Because the retrieval runs also materialize timeout transcripts and the arms are matched per round rather than at equal total budget, this is per-repair trace accounting, not a clean cost-efficiency comparison. Certificate graph features (explored size, acceptance rate, goal length) predict repairability with cross-validated AUC near 0.67, a first step toward using certificates to prioritize where budget is spent.

5. Relation to Existing Theorem-Proving Work

Lean and mathlib provide the formal substrate for large-scale verified mathematics (de Moura & Ullrich, 2021; The mathlib Community, 2020). Neural theorem-proving work has studied tactic prediction, proof-artifact co-training, retrieval, and benchmarks for Lean proof search (Han et al., 2021; Yang et al., 2023; Xin et al., 2025). Our goal is different. We do not train a prover or claim competitive theorem coverage. We study the audit layer around a proof agent: once a model has exposed a finite set of tactic candidates, what can the agent certify, repair, and acquire next without weakening the verifier?

This also connects to structured generation and tool use. Constrained decoding systems try to keep generated outputs valid (Scholak et al., 2021; Lu et al., 2022; Beurer-Kellner et al., 2023; Dong et al., 2024), and tool-using agents interleave language-model choices with external actions (Yao et al., 2023; Schick et al., 2023). The same finite-support optimization view also applies to structured tool calls, where a validator checks whether generated actions satisfy an external schema or execution contract. Lean gives the AI4Math-native version: the external validator is a proof assistant, and the accepted outputs are machine-checked proof paths.

6. Limitations

CSLib-Holes is derived from an external project but still modest relative to standard theorem-proving benchmarks. It is designed to test finite-support audit semantics. We do not run a standard external benchmark (MiniF2F, ProofNet, miniCTX) or a prior retrieval/repair system (e.g., COPRA, Baldur, premise selectors); anchoring CSLib-Holes against those is left to future work. The result depends on the fixed Lean version, CSLib commit, REPL behavior, imports, prompt, candidate budget, depth bound, timeouts, and retrieval corpus. Certificates are scoped to the recorded support. They do not prove theorem falsehood or absence of a proof outside the trace. The proof-state oracle-clean subset is conservative: 23 original CSLib proofs replay through our tactic-level extraction protocol. Command-level oracle replay broadens post-hoc taxonomy to 87/100 holes, but it checks restored declarations rather than proof-state tactic steps and remains oracle-only diagnostic evidence. The certificate-frontier experiment is oracle-free, but it is still a post-hoc support update over a frozen run rather than an online agent that decides when to retrieve during proof search. The target-only control absorbs much of the Round 5 retrieval gain, so the current evidence should not be read as a strong causal claim that frontier states uniquely drive the improvement. We also tested whether certificates could be reused as proof-state suffixes; the current artifacts gave 0/74 reconstructible solving suffixes and 0 nontrivial state matches, so this paper treats closed certificates as audit triggers, not as reusable proof fragments. A stronger version should improve CSLib proof extraction, add more external Lean projects, and test richer online support-acquisition policies.

7. Conclusion

Finite-support replay gives Lean proof agents a modest but useful failure semantics. On CSLib-Holes, the audit layer turns recorded tactic support into Lean-checked proofs where possible, closed finite-support certificates when the replay graph is exhausted, and resource-bound transcripts when timeouts occur. The oracle-clean subset shows why this matters: after exact replay, the remaining failures are not unverifiable model judgments, but replayable evidence that the current finite support lacks an accepted proof path.

The support-update claim is deliberately bounded. A closed certificate does not certify theorem falsehood. It identifies a place where the recorded support is exhausted. The positive result appears when a fixed automatic retrieval policy exposes additional declaration signatures and every new proof is replayed by Lean. On the 74 closed base certificates, certificate-frontier retrieval repairs 16 cases, target-only retrieval repairs 12, and no-retrieval wider replay repairs 6, every accepted repair Lean-checked. Unioning frontier retrieval with deterministic retrieved-signature tactics gives 18 distinct support updates and improves the fixed full-run closure from 10/100 to 28/100. Further support expansion over residual certificates adds verified repairs in two more

rounds; the strongest frontier/control composition reaches 38/100, and the legal oracle-free portfolio reaches 39/100 after adding the remaining target-only success. The contribution is not a claim that frontier retrieval uniquely explains the gains. It is a Lean-checked audit layer for mathematical agents, separating proved goals, exhausted recorded support, and support updates that genuinely help because Lean accepts the new proof.

References

Beurer-Kellner, L., Fischer, M., and Vechev, M. Prompting is programming: A query language for large language models. *Proceedings of the ACM on Programming Languages*, 7(PLDI):1946–1969, 2023. doi: 10.1145/3591300.

de Moura, L. and Ullrich, S. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pp. 625–635. Springer, 2021. doi: 10.1007/978-3-030-79876-5_37.

Dong, Y., Ruan, C. F., Cai, Y., Lai, R., Xu, Z., Zhao, Y., and Chen, T. XGrammar: Flexible and efficient structured generation engine for large language models. *arXiv preprint arXiv:2411.15100*, 2024. doi: 10.48550/arXiv.2411.15100. MLSys 2025.

Han, J. M., Rute, J., Wu, Y., Ayers, E. W., and Polu, S. Proof artifact co-training for theorem proving with language models. *arXiv preprint arXiv:2102.06203*, 2021. doi: 10.48550/arXiv.2102.06203.

Lu, X., Welleck, S., West, P., Jiang, L., Kasai, J., Khashabi, D., Le Bras, R., Qin, L., Yu, Y., Zellers, R., Smith, N. A., and Choi, Y. NeuroLogic A*esque decoding: Constrained text generation with lookahead heuristics. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 780–799. Association for Computational Linguistics, 2022.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. doi: 10.48550/arXiv.2412.15115.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023. doi: 10.48550/arXiv.2302.04761.

Scholak, T., Schucher, N., and Bahdanau, D. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 9895–9901. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.779.

The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pp. 367–381. ACM, 2020. doi: 10.1145/3372885.3373824.

Xin, R., Zheng, Z., Nie, Y., Yuan, K., and Xiao, X. Scaling up multi-turn off-policy RL and multi-agent tree search for LLM step-provers. *arXiv preprint arXiv:2509.06493*, 2025.

Yang, K., Swope, A. M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R., and Anandkumar, A. LeanDojo: Theorem proving with retrieval-augmented language models. *arXiv preprint arXiv:2306.15626*, 2023. doi: 10.48550/arXiv.2306.15626.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2023. doi: 10.48550/arXiv.2210.03629. ICLR 2023.

A. Artifact Paths

Code, the CSLib-Holes benchmark, and the audit checks are available at https://github.com/rfarell/cslib-holes. The CSLib-Holes benchmark is derived from the public CSLib project (Apache-2.0); the repository preserves that attribution. Within this artifact, the CSLib-Holes experiment lives under `experiments/cslib_holes_external`. The frozen manifests are `benchmark_manifest.json`, `cases.json`, `oracles.json`, `cases_oracle_clean.json`, and `oracles_oracle_clean.json`. Full result-directory names, oracle-gap audits, and retrieval packet metadata are listed in `experiments/cslib_holes_external/README.md` and `paper_tables/cslib_holes_results.json`. The one-command artifact check is `python experiments/cslib_holes_external/check_artifacts.py`; the paper-facing claim guard is `python paper/verify_ai4math_claims.py`. These checks validate summary counts, table rows, retrieval-context metadata, and oracle-gap audit summaries; they do not rerun all model acquisitions.